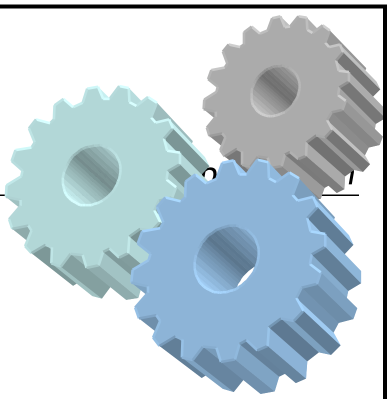


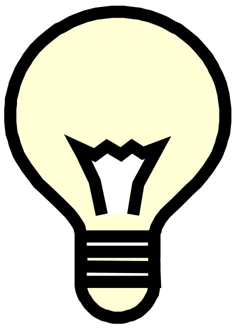
71% of the software projects do not succeed!

Top 10 Tips for Successful Software Development Management

by Jack Bicer

Here are some time tested guidelines that have been used extensively to deliver web development projects successfully, on-time and on-budget. Although most of the projects developed are between 1 to 10 man years of effort, these tips also scale nicely for smaller projects of 2 man months to larger projects of 25 man years.

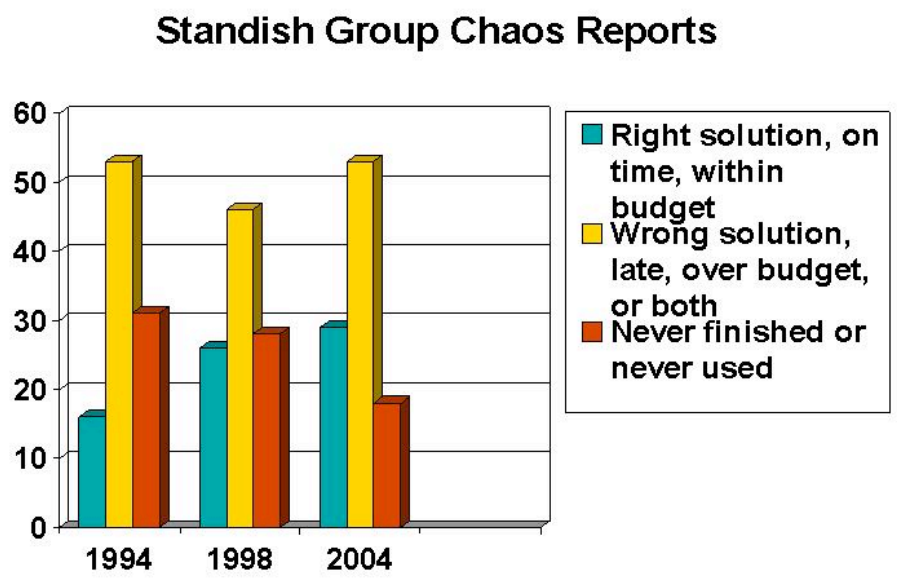
1. Understand the user’s needs, write the specs before coding and keep them up to date. Develop the User Interface with the specs and flush out design issues.
2. Break projects into modules of 1 week or shorter.
3. Implement risky modules early.
4. Create validation milestones, every 3 to 4 weeks.
5. Provide the necessary resources.
6. Get developer buy-in for features, timelines and milestones.
7. Keep people accountable to their commitments.
8. Resist “feature creep” during implementation and testing.
9. Use automated functional testing tools and do stress testing.
10. Under-promise, over-deliver and plan a pleasant surprise at the end.



In every project, one or more of these guidelines will be broken. A breach of a guideline does not mean a project will be unsuccessful. Other guidelines will usually pull you through and help the project succeed. It is the large number of breaches and the depth of the breaches that will create project failures.

Septium Corporation • Irvine, CA • eMail: info@septium.com • Phone: (714) 389-3360

According to the research data compiled by the Standish Group on over 50,000 projects from around the world, 71% of the software projects fail or are significantly challenged. This is an improvement over the 84% rate 10 years ago, but still leaves a lot of room for improvement.



In this whitepaper, we assume a competent development team, able to estimate the work. The fine art of estimation is left as a topic for another white paper.

Every project has one or more target audiences and the people representing these audiences. For the purposes of our projects, we usually refer to them as customers. Whether they are internal to your organization or external to your company, I like the consulting point of view that every project has a customer and our job is make our customers happy.

1. Understand needs, write the specs before coding and keep them up to date.

Once you understand the functionality to be built, you'll need to do a detail level design, preferably in terms of use cases. Work out the design details before the start of coding. Doing so during coding will increase the failure rate of the project. This does not mean you can not make design changes later, just try to minimize those changes by doing a great job upfront, as changes are the most significant factor contributing to project failures. If a change costs $1 to incorporate during the design phase, the same change will cost $10 during the coding phase and over $100 after the system is in production. An easy way to minimize software development costs is to do a great job during the design phase and capture the details on paper so everyone on the project team has the same expectations.

Develop the User Interface with the specs to flush out design issues. Once you lay out the elements of the user interface, your customers will see how this all comes together. If there are any issues with the design, it will come to light. The user interface design is also a great process to flush out or capture the details that may have fallen by the wayside.

Many projects utilize commercial components or code libraries that are new to the development team. I recommend developing a small test application demonstrating the team's ability to utilize the needed features in the commercial components. Other than the coding involved for such a test project, no other development should take place until the specs are completed. Although we like most of the Agile development concepts, the upfront documentation of the design specifications is one that we differ in opinion, especially for remote teams.

As you gain more insight and discover additional information that needs to be remembered, update the specs. When you make design decisions or changes, update the specs. Our ultimate goal is to make the specs a project bible where we can find most of the answers. Also keep in mind that our real output is the software, not the technical documentation. The specs are there to make the process work smoother.

2. Break projects into modules of 1 week or shorter.

When you have an architecture in place, break the tasks into modules of 1 week or shorter to get good visibility and predictability. For modules that are hard to break down, you can extend the timeframe up to 3 weeks. Most of the web modules we develop are usually 3 to 5 days in duration. Modules that take longer to implement are harder to manage and feature creep usually comes in, effecting deliverables.

3. Implement risky modules early.

Build modules that have some risk element, up front. If the project is going to fail, it may as well fail at the beginning, before you invest a lot of time, energy and resources. Try to understand what you don't know as early as possible. The further away the milestones, the more unknowns there are. If you minimize the risks and the unknowns up front, when new things come up, you'll be able to incorporate them into the project more easily simply because you left the easy stuff to the end.

4. Create validation milestones every 3 to 4 weeks.

Milestones are where "the rubber meets the road." You want to be able to demonstrate the milestones for each piece of functionality, show that they are working, share them with your customers and get their feedback. If you missed the boat, you want to find out early. Milestones tell your customers that you are doing the right things, at the right time and the project is likely to be a success. It reinforces where you are on the project plan. Milestones also force programmers to focus on the deliverables and get to the minor details that are often left until the end and often forgotten. A milestone demo will usually flush out any issues, known, hidden or unknown. What we don't know that we don't know will usually come to hurt us. Use milestones to get everyone involved on the same page and prove your progress. Milestones scheduled 3-4 weeks apart are usually a good way to show progress and catch issues before they become bigger problems.

5. Provide the necessary resources.

NEVER start a project without knowing that you are going to be able to finish it. The people assigned to the project will not take ownership of their tasks if they don't have the resources to finish them successfully and they will blame the management for that.

6. Get developer buy-in for features, timelines and milestones.

This is an important but often overlooked issue. Knowing the details of what you are building is one the most important factors in estimating accurately. After the design specifications are done, whenever possible, have the developers establish the coding timelines, rather than dictate the timelines to the developers. The developer is the one who understands how complicated the project is and how long it will take to complete. After all, they are the ones writing the code. If too much time is being requested by the developer, you can always ask questions, like "Why do you think this will take so long? Where do you see the problem with this project?". Understand the issues, flush out the details (which usually solves the problem) and treat issues fairly to both side's satisfaction in a friendly, fun conversations. You want to be able to hold developers accountable for the work that they are doing. In order to do that, you need to have everyone agree to his or her commitments wholeheartedly and establish clear expectations. Get their buy-in for the milestones and the timelines before they start coding.

7. Keep people accountable to their commitments.

This important point requires that you get the buy-in from the developers, testers and subject matter experts since you are going to keep them accountable. This doesn't mean that commitments or timelines can never be violated, but it does mean that there needs to be a very good reason to break a commitment. If it is the developers' agreed upon timeline that is violated, they can be held accountable. If the violated timeline was set by the management, the developers' common excuse is that they tried, but the timeline was unreasonable or say "I didn't make the commitment, you did", whereby shifting the accountability. At the time the agreement is made, make sure that everyone is clear that they will be kept accountable to their commitments. Also be clear that everyone is expected to inform you when there is a good chance that they may not be able to keep their commitments. This allows you to manage by exception, giving you the ability to manage more projects and fewer surprises. Many times a problem caught early on can be mitigated so that it does not affect the deliverables.

8. Resist feature creep during implementation and testing.

CAUTION, HARMFUL IF SWALLOWED! This is by far the worst enemy of the projects, especially in new applications where Version 1.0 is being built. The customers will think of additional features or different ways of doing things during the project implementation or the testing phase, well after the design is completed and approved. We would recommend starting a new list for these requests, to be implemented as an add-on phase to the project, as soon as the project completes the testing phase. Resist adding new functionality unless it is trivial to implement (i.e. without effecting project timelines) or is detrimental to the success or the completion of the project. Be supportive but use your best judgment with the understanding that feature creep is a very slippery slope. Although hard to do, this is probably one of the most effective tips we can give you. The question to ask is "Do you want high quality software, with the desired features, on time and on budget, or do you want the flexibility of changing the feature set when you want, at a significantly increased cost and unpredictability?". If you get a buy-in from the customer and the executive management team *before* the start of the project, this is doable.

9. Use automated functional testing tools and do stress testing.

Create a test plan while the code is being developed. It's important to know what to test and how to test it, before starting the testing phase. This can be done on the surface, without going into much detail or it can be a very detailed test plan, depending on the amount of time and effort you want to spend on quality assurance. Unfortunately there is always a trade-off between high quality software with the minimal amount of bugs and the amount of money/resources/time invested in testing. You should find out where this trade-off is for your organization and do what is normally expected, unless it is broken.

Use automated testing tools. These are tools that make repetitive testing fast, accurate and a lot easier. Once the test scripts are developed, the entire system can easily be retested whenever a new feature is added, just to make sure nothing has been broken unexpectedly.

Web applications will also need to go through stress testing, especially when the expected user base is over 1,000 users. Although the usage patterns and application bottlenecks differ from application to application, every new web application has at least one "Achilles Heel" that will bring the system to it's knees. Test your system ahead of time and know what your limits are; every application has them. Today, servers are inexpensive and powerful; many performance issues can be solved just by adding extra servers. If you don't know where your bottlenecks are, you will not know when to bring in additional resources. Know your limits, accept them or fix them.

10. Under promise, over deliver and plan a pleasant surprise at the end.

This adage is heard and known by almost all of us yet seldom put to practice. To your customers, always promise just a little less than what you can deliver. Internally, always push to deliver a little bit more than what you can deliver. I recommend keeping two separate project plans, one for development and one for your customer. Although this may seem a bit inefficient, and you may get eyebrows raised for not running the software development at maximum effectiveness, remember that software development is still an art, surprises will almost always happen and what people will remember is how you finished the project. *Successfully or unsuccessfully!*

Murphy's Law will present itself, something unexpected will happen, so at worst you want to be able to deliver what's promised and maybe something slightly better. At best, you'll either deliver additional desired features or deliver the project in less time, under budget, and really score big brownie points. That works for both the business side and the technical side of the house simply because IT is now seen as a "*trusted partner*" in delivering the needed solutions on time, on budget. The business side of the house is usually very happy because they get predictable results. Your developers are happy, because they are now seen as the heroes.

Proper expectation management is vital to the success of the project. Good project managers will spend a lot of time in managing expectations by under promising, over delivering and especially planning a pleasant surprise at the end. This makes the projects look more successful.

About the author:

	Jack Bicer is the CEO & CTO of Septium Corp., a *hybrid* offshore custom web software development company, combining offshore cost savings with US management expertise. Prior to Septium, Jack held several CTO roles in Internet companies since the early days of the Internet and as a turnaround CTO, enjoyed the sale of 1GlobalPlace to VeriSign. A 28 year software industry veteran, Jack is an industry expert in SaaS/web software, new product development, off-shoring, technology strategy and management. Known as the "*Father of Uninstall*", he invented the uninstall concept and wrote the first uninstaller. He is also credited with inventing the "Automated Software Updates". In 2003, Jack founded TechBiz Connection, one of the largest non-profit technology management associations in Southern California, and currently serves as its Chairman and President. Jack can be reached at jbicer@septium.com.